Natural Language Processing 2: Deep learning, transformer models

HS Sprachtechnologie für eine bessere Welt (Winter semester 2021/22)

Nils Reiter,
nils.reiter@uni-koeln.de

November 2, 2021

Today: Automatization

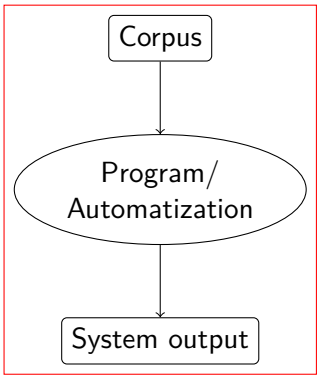


Table of Contents

Task Types and Methods

Task	Method(s)
Classification	Decision tree, support vector machine, neural network, naïve Bayes, k-nearest neighbors, ...
Clustering	k-means, affinity propagation, ...

Task Types and Methods

Task	Method(s)
Classification	Decision tree, support vector machine, neural network, naïve Bayes, k-nearest neighbors, ...
Clustering	k-means, affinity propagation, ...
Ranking	Ranking SVM, PageRank, ...
Sequence labeling	Hidden Markov models, conditional random field, neural networks ...
Segmentation	...

Supervised Machine Learning

Two parts to understand

Prediction Model

How do we make predictions on data instances?
(e.g., how do we assign a part of speech tag to a (unlabeled) word?)

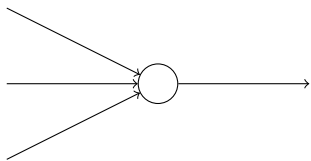
Learning Algorithm

How do we create a prediction model, given annotated data?
(e.g., how do we create rules for assigning a part of speech tag to a word?)

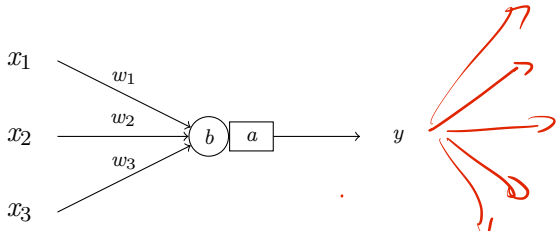
Section 1

Neural Networks

A Neuron



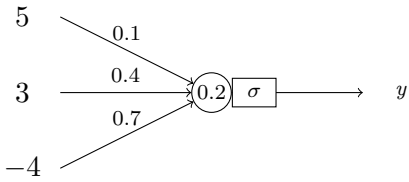
A Neuron



$$y = a(w_1 x_1 + w_2 x_2 + w_3 x_3 + b)$$

A Neuron

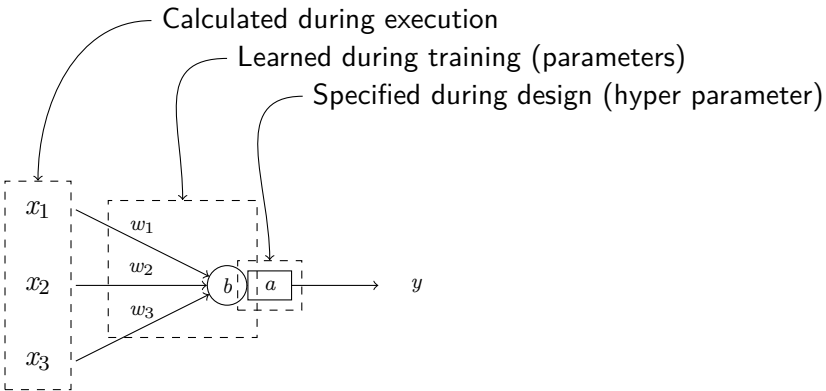
Example



$$
\begin{aligned}
y &= a(w_1 x_1 + w_2 x_2 + w_3 x_3 + b) \\
&= \sigma(0.1 \times 5 + 0.4 \times 3 + 0.7 \times -4 + 0.2) \\
&= \sigma(-0.9) \\
&= 0.28905049737499603653 69
\end{aligned}
$$

A Neuron

Where do these values come from?



Many Neurons make a Network

Notation

w_{jm}^{kn}: Connection between neuron j in layer k and neuron m in layer n

a: activation function

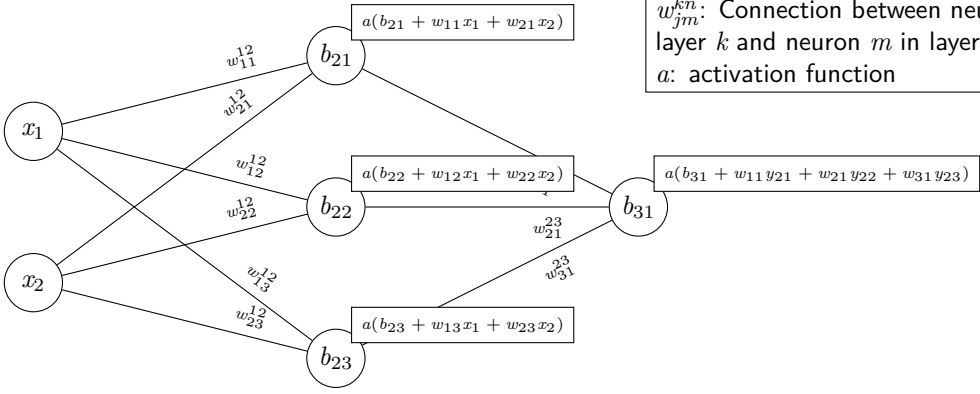


Figure: A simple neural network with 1 hidden layer

Forward Pass

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions to calculate individual values in sequence

Forward Pass

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions to calculate individual values in sequence
- Practically, a lot of the computation happens in matrices in parallel
  - Hidden layer
    - Weights: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = (b_{21}, b_{22}, b_{23})$

Forward Pass

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions to calculate individual values in sequence
- Practically, a lot of the computation happens in matrices in parallel
  - Hidden layer
    - Weights: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = (b_{21}, b_{22}, b_{23})$
- Hidden layer computation: $f_2(X) = \sigma(\underbrace{W_{1,2}^{\intercal} X + B_2}_{\text{matrix operations}})$

Forward Pass

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions to calculate individual values in sequence
- Practically, a lot of the computation happens in matrices in parallel
  - Hidden layer
    - Weights: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = (b_{21}, b_{22}, b_{23})$
- Hidden layer computation: $f_2(X) = \sigma(\underbrace{W_{1,2}^{\intercal} X + B_2}_{\text{matrix operations}})$
- Deep learning involves a lot of matrix operations
  - GPUs are highly optimized for this
  - Hint: Gaming-GPUs that support CUDA are also usable for deep learning

Feed-Forward Neural Networks

- The above is called a "feedforward neural network"
  - Information is fed only in forward direction

Feed-Forward Neural Networks

- The above is called a "feedforward neural network"
  - Information is fed only in forward direction
- Configuration/design choices
  - Activation function in each layer
  - Number of neurons in each layer
  - Number of layers

Processing Language

- Neural networks operate on numbers
- To process language, we need to preprocess our data

Processing Language

- Neural networks operate on numbers
- To process language, we need to preprocess our data

Word Indices

1. Establish the vocabulary (i.e., the set of all known tokens [in the training corpus])
2. Create a ranking (i.e., count all word types)
3. Decide on a threshold (e.g., the $10\,000$ most frequent words)
4. Replace all words above the threshold by an index number
5. Replace all other words by a special symbol

Processing Language

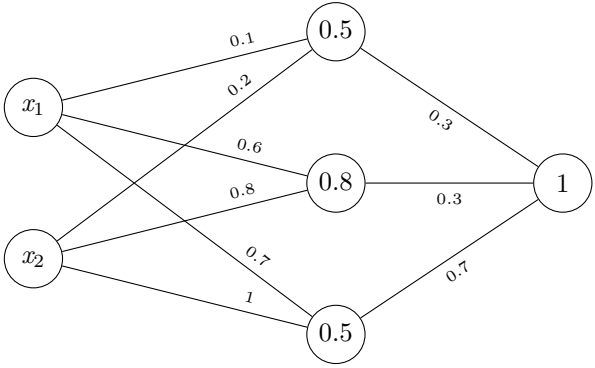
- Neural networks operate on numbers
- To process language, we need to preprocess our data

Word Indices

1. Establish the vocabulary (i.e., the set of all known tokens [in the training corpus])
2. Create a ranking (i.e., count all word types)
3. Decide on a threshold (e.g., the $10\,000$ most frequent words)
4. Replace all words above the threshold by an index number
5. Replace all other words by a special symbol

$\Rightarrow$ "Out of vocabulary" (OOV) words are a challenge for applications

Example



x_1	x_2	y
0	0	0.86169636
1	0	0.87786007
1	1	0.891605
10	10	0.90814614
⋮	⋮	⋮

Figure: Neural network with randomly initialized weights

```
5  # setup the model architecture
6  model = keras.Sequential()
7  model.add(layers.InputLayer(input_shape=(2,)))
8  model.add(layers.Dense(3, activation="sigmoid"))
9  model.add(layers.Dense(1, activation="sigmoid"))
10
11 model.compile() # compile it
12
13 w1 = [ # weights between neurons
14   np.array([[0.1,0.6,0.7],[0.2,0.8,1]]),
15   # bias terms
16   np.array([0.5,0.8,0.5]) ]
17
18 w2 = [ # weights between neurons
19   np.array([[0.3],[0.3],[0.7]]),
20   # bias terms
21   np.array([1]) ]
22 model.layers[0].set_weights(w1)
23 model.layers[1].set_weights(w2)
24
25 y = model.predict(np.array([[0,0]])) # generate predictions
26 print(y)
```

Neural network with manually specified weights as above
Ilias: `simple-nn.py`

Learning Algorithm

- We can immediately calculate outcomes (= make predictions), even if all weights are generated randomly
- How do we improve the weights?

Learning Algorithm

- We can immediately calculate outcomes (= make predictions), even if all weights are generated randomly
- How do we improve the weights?
- Gradient Descent
  1. Initialize all weights randomly
  2. Calculate and derive the loss (the 'wrongness') of the current weights on the training data
  3. Check if we have found the optimal solution
  4. If not, calculate the direction in which the loss decreases
  5. Go back to 3.

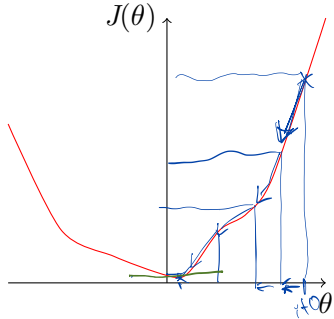
Section 2

Gradient Descent

Loss function: Intuition

- Loss should be as small as possible
- Total loss can be calculated for given parameters θ
  - θ is a vector containing all weights and bias terms in the network
- Idea:
  - We change θ until we find the minimum of the function
  - We use the derivative to find out if we are in a minimum
  - The derivative also tells us in which direction to go

Loss function: Intuition



Loss and Hypothesis Function

- Hypothesis function h
  - Calculates outcomes, given feature values x
  - Done by the neural network
- Loss function J
  - Calculates 'wrongness' of h, given parameter values θ (and a data set)
  - In reality, θ represents millions of parameters

Loss function: Definition

- Different loss function are in use
- Which one to use depends on our aims

Binary Cross-Entropy Loss

- Loss function used for binary classification problems
- Assumption: Output of the network is in $[0; 1]$, $0/1$ representing the two classes

$$J(\theta) = -\frac{1}{m} \sum_{i=0}^{m} y_i \log h_\theta(x_i) + (1 - y_i) \log(1 - h_\theta(x_i))$$

Loss function: Definition

Binary Cross-Entropy Loss

$$J(\theta) =$$

- m Number of training instances
- y_i The true outcomes (from training data)
- x_i The input values

Loss function: Definition

Binary Cross-Entropy Loss

$$J(\theta) = -\frac{1}{m}\sum_{i=0}^{m}$$

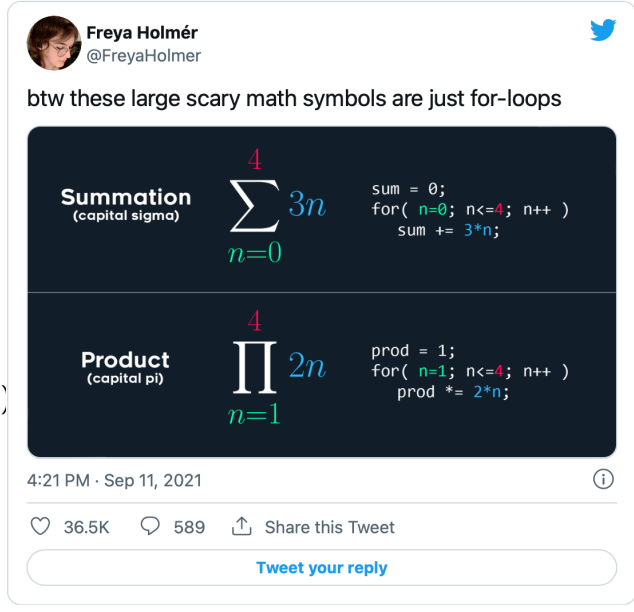
- m Number of training instances
- y_i The true outcomes (from training data)
- x_i The input values

Loss function: Definition

Binary Cross-Entropy Loss

$$J(\theta) = -\frac{1}{m}\sum_{i=0}^{m}$$

- m Number of training instances
- y_i The true outcomes (from training data)
- x_i The input values



Loss function: Definition

Binary Cross-Entropy Loss

$$J(\theta) = -\frac{1}{m}\sum_{i=0}^{m} \underbrace{y_i \log_2 h_\theta(x_i)}_{0 \text{ iff } y_i=0}$$

- m Number of training instances
- y_i The true outcomes (from training data)
- x_i The input values

Loss function: Definition

Binary Cross-Entropy Loss

$$J(\theta) = -\frac{1}{m}\sum_{i=0}^{m} \underbrace{y_i \log_2 h_\theta(x_i)}_{0 \text{ iff } y_i=0} + \underbrace{(1-y_i)\log_2(1-h_\theta(x_i))}_{0 \text{ iff } y_i=1}$$

- m Number of training instances
- y_i The true outcomes (from training data)
- x_i The input values

Summary

- Neural network consists of layers of neurons
- Training goal: Find weights, such that the training instances are correctly predicted
- Training method: Gradient descent

Summary

- Neural network consists of layers of neurons
- Training goal: Find weights, such that the training instances are correctly predicted
- Training method: Gradient descent
- Training does not have to be completed in one go
  - Pausing at any time is possible
  - Training can continue with a different data set

Section 3

Encoder-Attention-Decoder Architecture

Different Layer Types

- So far: fully connected layer
- Other layers
  - Convolutional layer
  - Dropout layer
  - Recurrent layer
  - Long short-term memory (LSTM) layer
  - ...

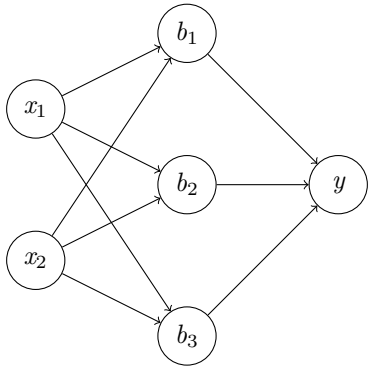
Different Layer Types

- So far: fully connected layer
- Other layers
  - Convolutional layer
  - Dropout layer
  - Recurrent layer
  - Long short-term memory (LSTM) layer
  - ...

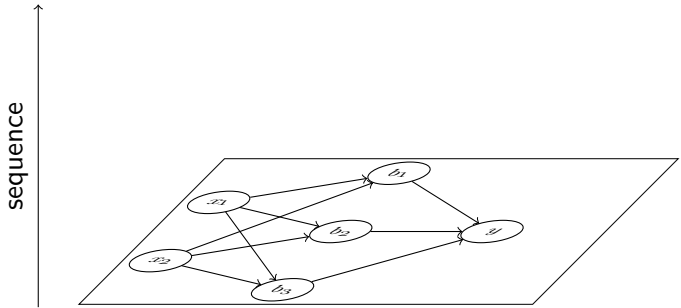
Sequences are important for NLP

- Many NLP tasks are sequential tasks: The outcome of one item has impact on the next item (e.g., part of speech)
- Recurrent and LSTM layers add new connections
- Instead of processing one item at a time, they look at sequences
- Connections along the sequence (i.e., the neuron knows its output for the previous item)

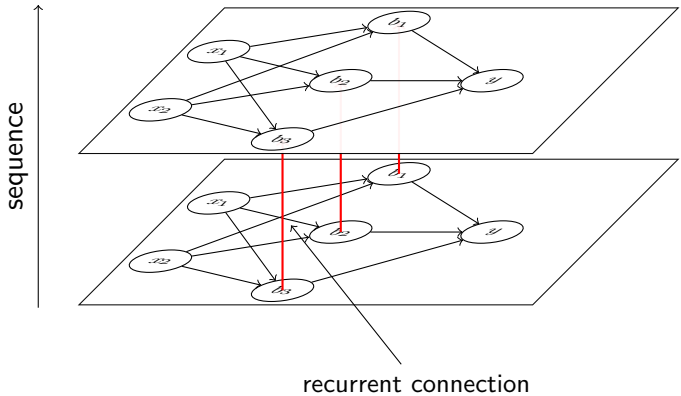
Recurrent Neural Networks



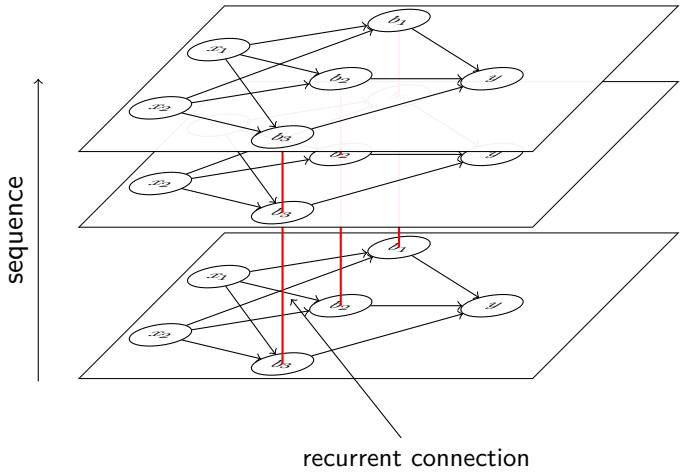
Recurrent Neural Networks



Recurrent Neural Networks



Recurrent Neural Networks



Recurrent Neural Networks

- Feed-forward neural networks: Weights between neurons
- Recurrent neural networks
  - Weights between neurons
  - Weight(s) for recurrent connections

Recurrent Neural Networks

- Feed-forward neural networks: Weights between neurons
- Recurrent neural networks
  - Weights between neurons
  - Weight(s) for recurrent connections
- Also possible in two directions

Recurrent Neural Networks

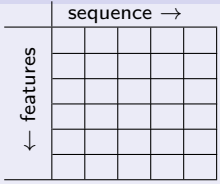
- Feed-forward neural networks: Weights between neurons
- Recurrent neural networks
  - Weights between neurons
  - Weight(s) for recurrent connections
- Also possible in two directions

Input shape

RNN layers need 2D input:

- Input sequences (if needed, padded to the same length)
- Features (e.g., embeddings)

Training on multiple sequences then requires 3D input!



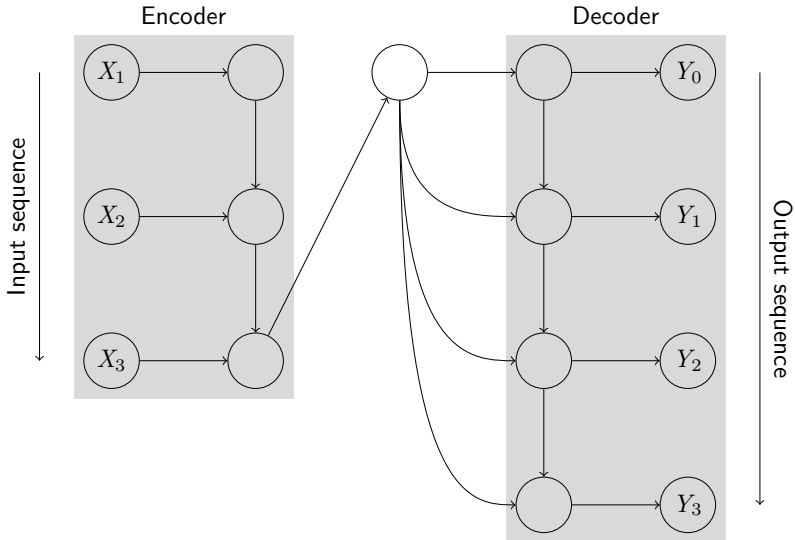
Encoder-Decoder-Architecture

- Often: No 1-to-1 relation between input and output
  - I.e.: Not as many output items as input items (e.g., machine translation)

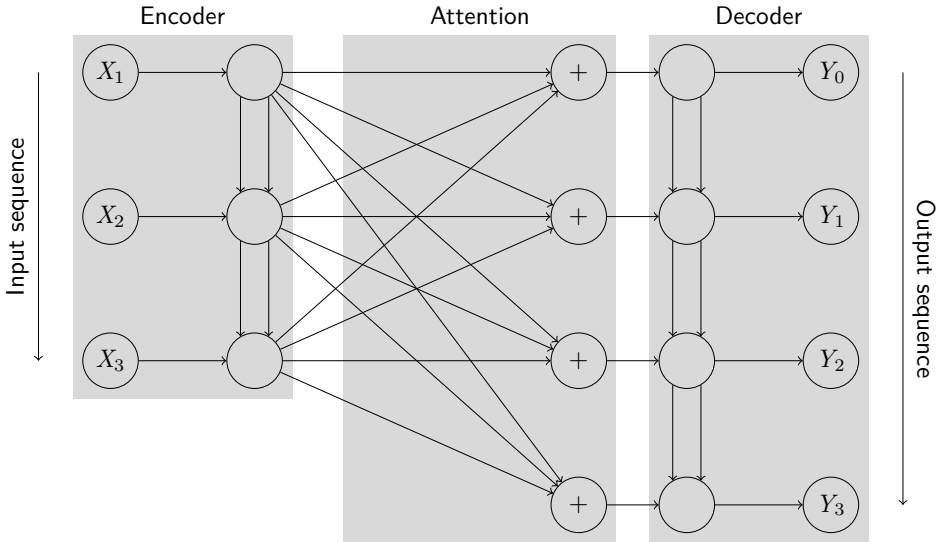
Encoder-Decoder-Architecture

- Often: No 1-to-1 relation between input and output
  - I.e.: Not as many output items as input items (e.g., machine translation)
- Encorder-decoder-network has two parts:
  - Encoder maps from input data to an internal representation
  - Decoder maps from internal representation to the output
- Internal representation
  - Use the output or internal state of last recurrent cell
  - Not interpretable

From Encoder-Decoder to Attention



From Encoder-Decoder to Attention



Section 4

BERT

Introduction

- BERT has outperformed the state of the art in many tasks
- Breakthrough in natural language processing

Introduction

- BERT has outperformed the state of the art in many tasks
- Breakthrough in natural language processing
- General idea
  - Encoder-Attention-Decoder architecture (= transformer)
  - Process whole input at once (max. 512 tokens, = bidirectional)
  - Pre-training and fine-tuning on different tasks

Jacob Devlin et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423

Pre-Training and Fine-Tuning

- BERT models are trained on huge data sets
- Training one from scratch requires significant resources (time/money)
- Pre-trained models are shared freely
- Recipe: Take a pre-trained model and fine-tune it on your task
  - Pre-trained model contains an abstract language representation

Pre-Training and Fine-Tuning

- BERT models are trained on huge data sets
- Training one from scratch requires significant resources (time/money)
- Pre-trained models are shared freely
- Recipe: Take a pre-trained model and fine-tune it on your task
  - Pre-trained model contains an abstract language representation
- Fine-tuning
  - Any language-related task!

BERT Training Tasks

Masked Language Modeling (MLM)

- Sentence-wise
- 15% of the tokens are “masked” by a special token
- Model predicts these, having access to all other tokens

BERT Training Tasks

Masked Language Modeling (MLM)

- Sentence-wise
- 15% of the tokens are "masked" by a special token
- Model predicts these, having access to all other tokens

Next sentence prediction (NSP)

- Two (masked) sentences are concatenated
- Model has to predict wether second sentence follows on the first or not

Section 5

Summary

Summary

Neural networks

- Consist of neurons, which combine values from previous neurons
- Matrix computation
- Can ‘learn’ any relation between input and output

Summary

Neural networks

- Consist of neurons, which combine values from previous neurons
- Matrix computation
- Can ‘learn’ any relation between input and output

Gradient descent

- Basic form to train a neural network
- Start with random weights, then iteratively improve
- Loss: Quantification of the wrongness of the current weights

Summary

Neural networks

- Consist of neurons, which combine values from previous neurons
- Matrix computation
- Can 'learn' any relation between input and output

Gradient descent

- Basic form to train a neural network
- Start with random weights, then iteratively improve
- Loss: Quantification of the wrongness of the current weights

BERT

- Transformer architecture
- Pre-trained on MLM and NSP (NSP has received some criticism)
- Fine-tuning on specific NLP tasks
- Model has learnt something about language (but we cannot exactly say what)

References I

Devlin, Jacob et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.